On the Correctness of Sliding Window Protocols

Jan L.A. van de Snepscheut

Computer Science Department
California Institute of Technology

Caltech-CS-TR-91-06

On the Correctness of Sliding Window Protocols

Jan L.A. van de Snepscheut
Computer Science
California Institute of Technology
Pasadena, CA 91125

In this note some struggles with the sliding window protocol and the special case known as the alternating bit protocol, are reported. We try to give a correctness proof, and discover that we cannot do so for one of the versions of the sliding window protocol. One may either require channels that satisfy stronger assumptions or, as we will do, adapt the protocol and stick to the weaker assumptions. The alternating bit protocol can be traced back to [Bartlett]. We have been unable to trace back the origins of the sliding window protocols; [Stenning] discusses one of the versions and lists networks using related protocols.

1. A faulty channel

A communication protocol is used to provide reliable transmission of data over a faulty communication channel that garbles, duplicates, or loses data. We consider the case in which data is transmitted in one direction over the faulty channel, and we assume the presence of a channel in the opposite direction in order to be able to communicate the need for retransmission of a message. The latter channel is also faulty. No assumptions on the slack of the faulty or of the fault-free channel are to be made. It is assumed that a faulty channel operates as follows:

- messages arrive in the order in which they are sent;
- any message sent along the channel can be lost;
- any message sent along the channel can be duplicated;
- any message sent along the channel can be garbled; however, if a message is garbled this can be detected, i.e. the error detection mechanism is assumed to be perfect.
- the channel is not infinitely faulty, in the sense that only a finite number of messages can be lost or duplicated consecutively, and of the messages delivered only a finite number are garbled consecutively.

First, we give a program that implements a faulty channel. The program has input channel c and output channel d. The output on d is a faulty copy of the input on c, i.e. every message in d is accompanied by a boolean which indicates whether the message is garbled. We use functions *flip* and *flip'* which return a boolean value. They make a fair (but not necessarily random) choice between *true* and *false*.

$$*[c?x;\ *[flip \rightarrow d!(x, flip')]]$$

If the inner loop is iterated zero times then a message is lost; if it is iterated more than once then a message is duplicated. The *flip'* that occurs as an argument with x in the output command corresponds to the possibility of garbling the message. The choices made by *flip* and *flip'* are independent, both of them are fair.

Second, we show that we can restrict ourselves to loss and duplication of messages, i.e. we may assume that no messages are garbled. The reception of a garbled message does not give any information at the receiving side, it cannot even be concluded that a new message was sent. Hence, the only sensible thing that can be done with a garbled message is to ignore it. Therefore, we propose to use the faulty channel only in conjunction with a program that filters out all garbled messages.

$$\begin{array}{ll} & *[c?x;\ *[\mathit{flip} \rightarrow d!(x, \mathit{flip}')]] \\ \| & *[d?(y,b);\ [b \rightarrow \mathit{skip} [] \neg b \rightarrow e!y]] \end{array}$$

The latter combination is equivalent to

$$*[c?x;\ *[\mathit{flip} \rightarrow e!x]]$$

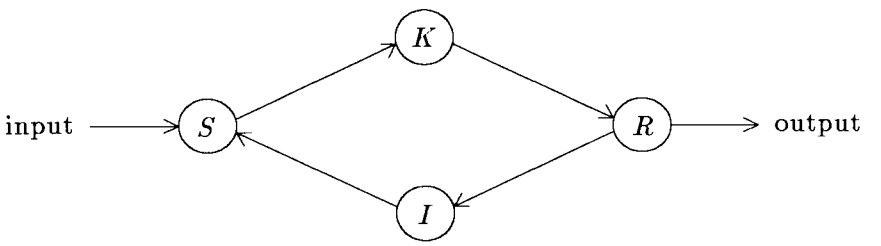
except for the slack in the communication, which is something that we want to ignore anyway. This program can also be written as

$$\begin{array}{l} c?x; \\ *[\mathit{true} \rightarrow c?x [] \mathit{true} \rightarrow e!x] \end{array}$$

where the choice between the two alternatives is assumed to be fair. This is the version of the faulty channel that we work with. It has the advantage that garbling of messages plays no role. The symmetry in input and output, apart from initialization, is also pleasing.

2. The alternating bit protocol

Next we consider the alternating bit protocol. We have a network as indicated in the figure below. Channels K and I are faulty channels, and the task is to construct programs S and R that, together, implement a fault-free communication line from input to output. We first consider a solution in which messages are numbered from 0 on and subsequently refine it to what is known as the alternating bit protocol by reducing the integers modulo 2. It is easier to do it this way rather than starting with the latter program because we would then have to introduce these integers anyway for proving the program's correctness.



The program text for the four processes is as follows. The channel from S to K is identified as channel sk, and similar for the other three channels. Channels sk and kr carry pairs, viz. an integer plus a message, and channels ki and is carry integers only.

$$
\begin{array}{ll}
K: & sk?(k,v); \\
 & *[true \rightarrow sk?(k,v) [] true \rightarrow kr!(k,v)] \\
I: & ri?i; \\
 & *[true \rightarrow ri?i [] true \rightarrow is!i] \\
R: & j := 0; \\
 & *[[\overline{ri} \rightarrow ri!j \\
 & \quad [] \overline{kr} \rightarrow kr?(h,u); \\
 & \qquad [h = j+1 \rightarrow j := j+1;\ out!u \\
 & \qquad [] h \neq j+1 \rightarrow skip \\
 & \quad]] \quad] \\
S: & in?w;\ l := 1; \\
 & *[[\overline{sk} \rightarrow sk!(l,w) \\
 & \quad [] \overline{is} \rightarrow is?h; \\
 & \qquad [h = l \rightarrow l := l+1;\ in?w \\
 & \qquad [] h \neq l \rightarrow skip \\
 & \quad]] \quad]
\end{array}
$$

We establish the correctness of this program in three steps. First we show that some invariants are maintained; next we show that deadlock is excluded, and finally we prove progress. We assume that the program's environment is such that communications on channels in and out are not suspended indefinitely. We also assume that the choice between the two guarded commands in S and R is fair if both probes are $true$. The invariant that we postulate is

$$
\begin{array}{l}
i \leq j \leq k \leq l \leq i+1 \quad \wedge \quad l = \#in \quad \wedge \quad j = \#out \quad \wedge \\
v = in_{k-1} \quad \wedge \quad w = in_{l-1} \quad \wedge \quad \langle \forall h : 0 \leq h < j : out_h = in_h \rangle
\end{array}
$$

where $\#in$ is the number of communications that have been completed on channel in and where in_h is message h transmitted along channel in, counting from 0 on. The invariant is established initially if we pretend the initial values of the variables to satisfy

$$
i = j = k = l = 0 \quad \wedge \quad v = w = in_{-1}
$$

for some fictitious value in_{-1}. We check that every assignment to one of the variables maintains the invariant.

- $sk?(k,v)$ matches $sk!(l,w)$ and together they are equivalent to $k,v := l,w$ which maintains the invariant;

- $ri?i$ matches $ri!j$ and together they are equivalent to $i := j$ which maintains the invariant;
- $kr?(h,u)$; $[h = j+1 \rightarrow j := j+1;\ out!u[]h \neq j+1 \rightarrow skip]$ matches $kr!(k,v)$ and together they are equivalent to $[k = j+1 \rightarrow j := j+1;\ out!v[]k \neq j+1 \rightarrow skip]$ which maintains the invariant;
- $is?h$; $[h = l \rightarrow l := l+1;\ in?w[]h \neq l \rightarrow skip]$ matches $is!i$ and together they are equivalent to $[i = l \rightarrow l := l+1;\ in?w[]i \neq l \rightarrow skip]$ which maintains the invariant;
- $in?w$; $l := 1$ is executed with precondition $l = 0$ and, hence, $i = 0$; it maintains the invariant.

Next we show the absence of deadlock. S and R can be suspended on $in?$ and $out!$ respectively, but we have assumed that those suspensions are temporarily only, or on the choice between a communication with K and I. We show that the two link processes eventually perform a communication. For example, K can be suspended on $sk?$ but this matches with the guarded command $\overline{sk} \rightarrow sk!(l,w)$ in S and the latter is part of a selection which is assumed to be fair. The other three channels are similar: no communication is suspended indefinitely.

Finally we show that the program makes progress. Observe that none of the four variables i, j, k, and l decreases. We show that their sum increases eventually. On account of the invariant $i \leq j \leq k \leq l \leq i+1$ we can distinguish four cases:

- $i < j = k = l = i+1$:

  $ri?i$ is eventually executed; since it matches $ri!j$ it follows that eventually i increases;

- $i = j < k = l = i+1$:

  $kr?(h,u)$; $[h = j+1 \rightarrow j := j+1;\ out!u[]h \neq j+1 \rightarrow skip]$ is eventually executed; since it matches $kr!(k,v)$ it follows that eventually j increases;

- $i = j = k < l = i+1$:

  $sk?(k,v)$ is eventually executed; since it matches $sk!(l,w)$ it follows that eventually k increases;

- $i = j = k = l < i+1$:

  $is?h$; $[h = l \rightarrow l := l+1;\ in?w[]h \neq l \rightarrow skip]$ is eventually executed; since it matches $is!i$ it follows that eventually l increases.

This completes the correctness proof of the algorithm that transmits messages and integer numbers. Because of $i \leq j \leq k \leq l \leq i+1$ all the integers involved differ by at most one and, hence, they can be reduced modulo two: the conditions $k = j+1$ and k **mod** $2 \neq j$ **mod** 2 are equivalent, and so are $i = l$ and i **mod** $2 = l$ **mod** 2. The program can then be written with booleans instead of integers by introducing booleans bl and bj that satisfy

$bl \equiv (l \textbf{ mod } 2 = 1)$ and $bj \equiv (j \textbf{ mod } 2 = 1)$. The program then reads as follows.

$$
\begin{array}{ll}
K: & sk?(bk, v); \\
 & *[true \rightarrow sk?(bk, v) [] true \rightarrow kr!(bk, v)] \\
I & ri?bi; \\
 & *[true \rightarrow ri?bi [] true \rightarrow is!bi] \\
R: & bj := false; \\
 & *[[\overline{ri} \rightarrow ri!bj \\
 & \quad [] \overline{kr} \rightarrow kr?(bh, u); \\
 & \qquad [bh \neq bj \rightarrow bj := \neg bj;\ out!u \\
 & \qquad [] bh = bj \rightarrow skip \\
 & \quad]] \quad] \\
S: & in?w;\ bl := true; \\
 & *[[\overline{sk} \rightarrow sk!(bl, w) \\
 & \quad [] \overline{is} \rightarrow is?bh; \\
 & \qquad [bh = bl \rightarrow bl := \neg bl;\ in?w \\
 & \qquad [] bh \neq bl \rightarrow skip \\
 & \quad]] \quad]
\end{array}
$$

3. The sliding window protocol

In the case of the alternating bit protocol, sender S and receiver R do not get very far out of step: the number of messages received via in exceeds the number of messages transmitted via out by at most one. For reasons of efficiency it is attractive to increase this slack from 1 to N, where N is a fixed positive integer. This generalization turns out to be rather tricky. We present two programs and show that one of them does not necessarily make progress. Both solutions are known in the literature as sliding window protocols. In both cases we return to the method of transmitting unbounded message numbers and reduce them modulo some constant later.

In our first program we change S and leave I, K, and R unaffected. The idea of this solution is to store in S not one but up to N messages that have been received via in and for which it has not yet been determined that they have been transmitted via out. (We postpone the problem of storing those messages in a bounded buffer, and use an unbounded buffer.) Besides l we introduce variables m and n, and the informal interpretation is

for all $h : 0 \leq h < l$: in_h has been transmitted via out,

for all $h : 0 \leq h < m$: in_h has been sent from S to K via sk,

for all $h : 0 \leq h < n$: in_h has been received by S via in.

Notice that l may be less than the number of messages actually transmitted via out : we merely have $l \leq \#out$. Similarly, m may be "too low". We see to it, however, that n is

not "too low", i.e. we maintain $n = \#in$.

$$
\begin{array}{lll}
S: & l, m, n := 0, 0, 0; & \\
 & *[[\overline{in} \wedge n < l + N & \rightarrow in?a(n);\ n := n + 1 \\
 & \ [] \overline{sk} \wedge m < n & \rightarrow sk!(m + 1, a(m));\ m := m + 1 \\
 & \ [] \overline{is} & \rightarrow is?l;\ m := \max(l, \textit{any number less than } m) \\
 &]] &
\end{array}
$$

The selection between the three guarded commands is assumed to be fair. The statement $m := \max(l, \textit{any number less than } m)$ deserves some explanation. It causes m to decrease if a low value for l has been received, which in turn triggers the retransmission of some messages. It is also this decrease that causes

$$l \leq i \leq j \leq k \leq m \leq n \leq l + N$$

not to be an invariant of the program. We show that the weaker

$$l \leq m \leq n \leq l + N \quad \wedge \quad l \leq i \leq j \leq n \quad \wedge \quad k \leq n$$

together with

$$
\begin{array}{l}
n = \#in \quad \wedge \quad j = \#out \quad \wedge \\
v = in_{k-1} \quad \wedge \quad \langle \forall h : 0 \leq h < j : out_h = in_h \rangle \quad \wedge \quad \langle \forall h : 0 \leq h < n : a(h) = in_h \rangle
\end{array}
$$

is invariant. The invariant is established initially if we pretend the initial values of the variables to satisfy

$$i = j = k = l = m = n = 0 \quad \wedge \quad v = in_{-1} \qquad .$$

We check that every assignment to one of the variables maintains the invariant.

- $sk?(k, v)$ matches $sk!(m + 1, a(m));\ m := m + 1$ and together they are equivalent to $k, v := m + 1, a(m);\ m := m + 1$ which maintains the invariant because of the guard $m < n$;
- $ri?i$ matches $ri!j$ and together they are equivalent to $i := j$ which maintains the invariant;
- $kr?(h, u);\ [h = j + 1 \rightarrow j := j + 1;\ out!u[]h \neq j + 1 \rightarrow skip]$ matches $kr!(k, v)$ and together they are equivalent to $[k = j + 1 \rightarrow j := j + 1;\ out!v[]k \neq j + 1 \rightarrow skip]$ which maintains the invariant;
- $in?a(n);\ n := n + 1$ maintains the invariant because of the guard $n < l + N$;
- $is?l;\ m := \max(l, \textit{any number less than } m)$ matches $is!i$ and together they are equivalent to $l := i;\ m := \max(l, \textit{any number less than } m)$ which maintains the invariant.

Next we show the absence of deadlock. We show that on each of the four channels a communication eventually takes place. K can be suspended on $kr!$ but this matches $\overline{kr} \rightarrow kr?$ in R and the latter is part of a selection which is assumed to be fair. Similar for channels ri and is. K can also be suspended on $sk?$ and this matches $\overline{sk} \wedge m < n \rightarrow sk!$ in S. We show that the situation where K is suspended on a communication via sk does not persist indefinitely. Observe that both l and n assume a nondecreasing sequence of values only; m may both increase and decrease. Assume that K is suspended on $sk?$. If the first guarded command is selected an unbounded number of times then n increases without bound. Since we have $n \leq l + N$, l increases without bound. Since we have $l \leq j$, j increases without bound. However, j increases only through a communication on kr and this is excluded because k is suspended on $sk?$. Hence, the first guarded command is eventually not executed any more, which implies that its guard is eventually *false*, i.e. $n = l + N$. The last of the three guarded commands may but need not decrease m. If it decreases m, it makes $m < n$ true, thereby enabling the middle guarded command. If it does not decrease m it establishes $l = m$ through the execution of $m := \max(l, \textit{any number less than } m)$. The middle guarded command is excluded from execution only if $m = n$ whenever the middle guard is selected. Because we eventually also have $n = l + N$ and $l = m$, we find that the middle guard is *false* whenever it is selected only if $l = m = n = l + N$, i.e. if $N = 0$. Since we have assumed $N > 0$, the middle guard is selected and found to be *true* eventually and, hence, a communication on channel sk occurs eventually. Thus we have shown the absence of deadlock.

Finally we turn to progress. We give a scenario to show that, despite the absence of deadlock, it is possible that no progress is made. The scenario is as follows. We assume $N = 2$. After two communications by S on in we have

$$i = j = k = l = m = 0 \quad \wedge \quad n = 2$$

and we proceed as follows.

- S and K communicate twice via sk, leading to $i = j = l = 0 \quad \wedge \quad k = m = n = 2$
- K and R communicate once via kr without changing the state
- R and I communicate once via ri without changing the state
- I and S communicate once via is leading to $i = j = l = 0 \quad \wedge \quad k = n = 2 \quad \wedge \quad (m = 0 \quad \vee \quad m = 1)$
- if $m = 1$ in the latter state then we postulate another communication between I and S via is leading to $i = j = l = m = 0 \quad \wedge \quad k = n = 2$
- S and K communicate twice via sk, leading to $i = j = l = 0 \quad \wedge \quad k = m = n = 2$

This is a state that we have seen before. Since all four channels have been involved in the cycle, the execution is fair. Informally speaking, the problem is that the same message, viz., the one carrying $a(0)$, is always lost by the faulty channel K. The message that is

not lost is not the next one "expected" by the receiver and is, therefore, discarded. This may seem unlikely but it is consistent with our (weak) assumption that the channel looses a finite number of consecutive messages only.

There are two possible solutions to this problem. One is to strengthen our assumptions on the channels' operation. The other is to stick to our weak assumptions and use a better algorithm. An algorithm that is better than the previous one is what is sometimes called selective retry. Instead of retransmitting all messages from l on, the receiver sends the numbers of the messages for which a (re)transmission is needed. This leads to the following solution. It is generally considered to be more efficient than the previous version. In fact, it is also more correct. The receiver stores messages arriving via K in a buffer of size W. We assume $W \geq 1$. The initial consecutive subsequence thereof is transmitted via *out* and the remaining ones are stored until the gaps are filled in. Receiver R sends the set of "missing" numbers via I to sender S. A message on channels ri and is is a set of numbers instead of a single number.

$$
\begin{array}{lll}
R: & j, r := 0, \emptyset; & \\
 & *[[\overline{out} \wedge j \in r & \rightarrow out!b(j);\ r := r \backslash \{j\};\ j := j+1 \\
 & \ [\!]\ \overline{ri} \wedge j \notin r & \rightarrow ri!\{j..j+W-1\} \backslash r \\
 & \ [\!]\ \overline{kr} & \rightarrow kr?(h,u); \\
 & & \quad [h \geq j \rightarrow r := r \cup \{h\};\ b(h) := u \\
 & & \quad [\!]\ h < j \rightarrow skip \\
 &]] & \quad] \\
S: & l, n, s := 0, 0, \{0\}; & \\
 & *[[\overline{in} \wedge n < l+N & \rightarrow in?a(n);\ n := n+1 \\
 & \ [\!]\ \overline{sk} \wedge s \cap \{l..n-1\} \neq \emptyset & \rightarrow m :\in s \cap \{l..n-1\};\ sk!(m, a(m)) \\
 & \ [\!]\ \overline{is} & \rightarrow is?s;\ l := \min(s) \\
 &]] &
\end{array}
$$

We show that

$$
\begin{array}{l}
\min(s) = l \leq j \leq n \leq l+N \ \wedge\ k < \min(n, j+W) \ \wedge\ \min(i) \leq j \ \wedge \\
n = \#in \ \wedge\ j = \#out \ \wedge\ \langle \forall h : 0 \leq h < j : out_h = in_h \rangle \ \wedge \\
v = in_k \ \wedge\ \langle \forall h : h \in r : b(h) = in_h \rangle \ \wedge\ \langle \forall h : 0 \leq h < n : a(h) = in_h \rangle \ \wedge \\
\emptyset \neq i \subseteq \{l..j+W-1\} \ \wedge\ r \subseteq \{j..\min(n, j+W)-1\} \ \wedge\ \emptyset \neq s \subseteq \{l..j+W-1\}
\end{array}
$$

is an invariant of this program. If we assume the initial values of the variables to be

$$
j = l = n = 0 \ \wedge\ k = -1 \ \wedge\ v = in_{-1} \ \wedge\ r = \emptyset \ \wedge\ i = s = \{0\}
$$

then the invariant holds initially since $W \geq 1$. We check that every assignment to one of the variables maintains the invariant.

- $sk?(k,v)$ matches $m :\in s \cap \{l..n-1\}$; $sk!(m,a(m))$ and together they maintain the invariant. The selection of a value m from the set $s \cap \{l..n-1\}$; $sk!(m,a(m))$ succeeds because the guard guarantees that the set is nonempty.
- $ri?i$ matches $ri!\{j..j+W-1\} \backslash r$ and together they maintain the invariant because of the guard $j \notin r$ and because $W \geq 1$.
- $out!b(j)$; $r := r \backslash \{j\}$; $j := j+1$ maintains the invariant because of the guard $j \in r$.
- $kr?(h,m)$; $[h \geq j \rightarrow r := r \cup \{h\}$; $b(h) := u[]h < j \rightarrow skip]$ matches $kr!(k,v)$ and together they maintain the invariant.
- $in?a(n)$; $n := n+1$ maintains the invariant because of the guard $n < l+N$.
- $is?s$; $l := \min(s)$ matches $is!i$ and together they maintain the invariant (because of $\min(i) \leq j$).

Next we show the absence of deadlock. We show that on each of the four channels a communication eventually takes place. Communications on channels kr and is are not suspended indefinitely because guards $\overline{kr}$ and $\overline{is}$ occur in fair selection statements. Communication on channel ri is guarded with $\overline{ri} \wedge j \notin r$. The latter conjunct becomes true eventually because communications on out are assumed to succeed and, hence, the command guarded with $\overline{out} \wedge j \in r$ leads to the increase of j. Since $\max(i) \leq j+W-1$ this can be done only a bounded number of times without intervening update of i, i.e. without intervening communication via ri. Communication on channel sk is guarded with $s \cap \{l..n-1\} \neq \emptyset$; since $l = \min(s)$ and hence $l \in s$, we have $s \cap \{l..n-1\} = \emptyset$ only if $l = n$. Furthermore, the first guarded command in the same selection statement is guarded with $\overline{in} \wedge n < l+N$ which succeeds unless $n = l+N$ (cf. discussion of previous algorithm). Hence, communication on channel sk is suspended indefinitely only when $l = n = l+N$, i.e. if $N = 0$. Since $N > 0$, communication on sk is not suspended indefinitely.

Finally we turn to progress. Observe that the variables j, l, and n do not decrease in value. We show that their value increases eventually. Because of the invariant $l \leq j \leq n \leq l+N$ we distinguish three cases. (They are not necessarily disjoint if $N > 1$.)

$n < l+N$

a communication on in eventually occurs, and this is accompanied by an increase of n.

$j < n$

We show that if $j \notin r$ then r is eventually extended with a new element. Since no element is removed from r and since the size of r is bounded we eventually have $j \in r$ in which case j increases. If $j \notin r$ then no element is removed from r and, hence, no element is added to $\{j..j+W-1\} \backslash r$; hence, s is eventually set to $\{j..j+W-1\} \backslash r$ and eventually k is an element of $\{j..j+W-1\} \backslash r$ and eventually this element is added to r.

$l < j$

eventually a communication on ri sets $\min(i)$ to j, and a subsequent communication on is increases l to $\min(i)$.

This shows that in each of the three cases at least one of the variables increases eventually, which proves that the algorithm makes progress.

We observe that the alternating bit protocol is indeed a special case of the sliding window protocol. Substituting 1 for N and W we find that all the sets involved are either empty or singleton sets. In the alternating bit protocol the loop in R is rewitten in such a way that $r = \emptyset$ is an invariant; wherever an element is added to r it is immediately removed again by choosing the alternative that outputs the element $b(j)$ on channel out and removes j from set r, restoring the latter to the empty set. Similarly for sender S. The main difference in the coding of the program is that message in_h sent from S via K to R is accompanied by number h in the sliding window protocol and by number $h + 1$ in the alternating bit protocol.

4. The range of k and j

Finally we turn to the reduction of sequence numbers for the case of the sliding window protocol. In the case of the alternating bit protocol we could reduce all numbers modulo 2, but in the present case the situation is slightly more complicated. (In fact, the proof in this section took me more time to construct than all the others combined.)

We introduce a number of variables who play a role in the proof only: set K that represents the set of all possible values that might have been chosen for k, and a handful of integers that are upper and lower bounds of the sets s, i, and K. The program is extended with all these variables as follows. Since we are concerned with the range of the sequence numbers, we omit the messages that are being transmitted and retain their

sequence numbers only.

$$
\begin{array}{lll}
K: & sk?(K, loK, hiK, k); & \\
 & *[true \rightarrow sk?(K, loK, hiK, k) [] true \rightarrow kr!k] & \\
I: & ri?(i, loi, hii); & \\
 & *[true \rightarrow ri?(i, loi, hii) [] true \rightarrow is!(i, loi, hii)] & \\
R: & j, r := 0, \emptyset; & \\
 & *[[j \in r & \rightarrow r := r \backslash \{j\};\ j := j + 1 \\
 & \ [] \overline{ri} \wedge j \notin r & \rightarrow ri!(\{j..j + W - 1\} \backslash r, j, j + W) \\
 & \ [] \overline{kr} & \rightarrow kr?h; \\
 & & \ [h \geq j \rightarrow r := r \cup \{h\} \\
 & & \ [] h < j \rightarrow skip \\
 &]] & \ ] \\
S: & l, n, s, los, his := 0, 0, \{0\}, 0, 1; & \\
 & *[[n < l + N & \rightarrow n := n + 1 \\
 & \ [] \overline{sk} \wedge s \cap \{l..n - 1\} \neq \emptyset & \rightarrow m :\in s \cap \{l..n - 1\}; \\
 & & \ sk!(s \cap \{l..n - 1\}, los, \min(his, n), m) \\
 & \ [] \overline{is} & \rightarrow is?(s, los, his);\ l := \min(s) \\
 &]] &
\end{array}
$$

We introduce $\overline{r}$ as the relevant part of the complement of set r and postulate the following invariant.

$$
\begin{array}{l}
\overline{r} = \{j..\infty\} \backslash r \ \wedge \ i \subseteq \{loi..hii - 1\} \ \wedge \ s \subseteq \{los..his - 1\} \ \wedge \\
k \in K \subseteq \{loK..hiK - 1\} \ \wedge \\
hii \leq loi + W \ \wedge \ his \leq los + W \ \wedge \ hiK \leq loK + \min(N, W) \ \wedge \\
loK \leq los \leq loi \leq j \leq hiK \leq his \leq hii \ \wedge \ hiK \leq n \ \wedge \\
\langle \forall x : x \geq hiK : x \in \overline{r} \rangle \ \wedge \\
\langle \forall x : x < hiK : x \in s \ \Rightarrow \ x \in K \rangle \ \wedge \\
\langle \forall x : x < his : x \in i \ \Rightarrow \ x \in s \rangle \ \wedge \\
\langle \forall x : x < hii : x \in \overline{r} \ \Rightarrow \ x \in i \rangle
\end{array}
$$

The invariant is established through execution of the first statement in each process. We check that every assignment to one of the variables maintains the invariant.

- $sk?(K, loK, hiK, k)$ matches

  $m :\in s \cap \{l..n - 1\};\ sk!(s \cap \{l..n - 1\}, los, \min(his, n), m)$

  and together they maintain the invariant since loK increases to los and hiK increases to $\min(his, n)$;
- $ri?(i, loi, hi)$ matches $ri!(\{j..j + W - 1\} \backslash r, j, j + W)$ and together they maintain the invariant since loi increases to j and, for $x < his$,

$$x \in \{j..j + W - 1\} \backslash r \ \Rightarrow \ x \in \overline{r} \ \Rightarrow \ x \in i \ \Rightarrow \ x \in s$$

- $r := r \backslash \{j\}$; $j := j + 1$ maintains the invariant since guard $j \in r$ implies that $\overline{r}$ does not change and since

$$j \in r \;\Rightarrow\; j \notin \overline{r} \;\Rightarrow\; j < hiK \quad \text{hence} \quad j + 1 \leq hiK$$

- $kr?h$; $[h \geq j \rightarrow r := r \cup \{h\} [] h < j \rightarrow skip]$ matches $kr!k$ and together they maintain the invariant since $k < hiK$ which implies that $\langle \forall x : x \geq hiK : x \in \overline{r} \rangle$ is maintained, and since adding k to r removes k from $\overline{r}$ which implies that $\langle \forall x : x < hii : x \in \overline{r} \;\Rightarrow\; x \in i \rangle$ is maintained;
- $is?(s, los, his)$; $l := \min(s)$ matches $is!(i, loi, hii)$ and together they maintain the invariant;

Since $hiK - loK \leq \min(N, W)$ it follows from $loK \leq j \leq hiK$ and from $loK \leq k < hiK$ that

$$-\min(N, W) < j - k \leq \min(N, W)$$

which implies that the sequence numbers can be reduced modulo $2 \cdot \min(N, W)$ in R. From $s \subseteq \{l..j + W - 1\}$ and $l \leq j \leq n \leq l + N$ it follows that all sequence numbers can be reduced modulo $N + W$ in S. Combining the two, it follows that all sequence numbers can be reduced modulo $N + W$ in the whole program.

5. Conclusion

We find it surprising how little attention has been paid to the correctness of the sliding window protocols, especially to the issue of progress.

In [Tanenbaum] we find hardly any correctness considerations; only the issue of using cyclic numbers is addressed (and only by example). Of the references, however, it is the only one that mentions the selective retry that turned out to be essential for progress.

In [Stenning] some safety properties are established (in an elegant way). We quote "however it is not shown that the protocol will progress".

In [Hailpern] it is shown that the alternating bit protocol satisfies both safety and progress requirements. It is shown that the sliding window protocol (without selective retry) satisfies safety requirements and makes progress. The proofs are given for a stronger channel, however, viz. "if the same message is sent over and over again, it will eventually be delivered (provided that the receiving process repeatedly accepts messages)". The difference with our weaker channels is that if repeatedly message A followed by message B is sent, then in our case it can only be guaranteed that every now and then a message arrives, possible only A's and never a B. Hailpern's stronger channels guarantee that both A and B arrive eventually. Although our scenario is unlikely if faulty behavior is random, it is not at all hard to construct a channel that looses every other message.

6. Acknowledgement

I am very grateful to Peter Hofstee and Johan Lukkien for refusing to be satisfied with my earlier arguments and forcing me to do better.

7. References

[0] K.A. Bartlett, R.A. Scantlebury, P.T. Wilkinson. A note on reliable full-duplex transmission over half-duplex links. *Communications of the ACM,* 12(5):260-261, May 1969.

[1] Brent T. Hailpern. Verifying Concurrent Processes Using Temporal Logic Lecture Notes in Computer science, volume 129, Springer Verlag, 1982.

[2] N. V. Stenning. A data transfer protocol *Computer Networks,* 1(2):99-110, September 1976.

[3] Andrew S. Tanenbaum. Computer Networks, second edition Prentice Hall, 1988.